DOCUMENT RESUME

ED 286 463 IR 012 795

AUTHOR Johanson, Roger P.
TITLE Educational Computing and Cognitive Skills: Curricular Issues and Prolog Prospects.
PUB DATE Apr 87
NOTE 27p.; Paper presented at the Annual Meeting of the American Educational Research Association (Washington, DC, April 20-24, 1987).
PUB TYPE Information Analyses (070) -- Viewpoints (120) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Cognitive Development; Cognitive Processes; *Instructional Effectiveness; *Programing; *Programing Languages; Research Problems

ABSTRACT

Following a summary and critique of the research on the use of computers in education to develop higher-order thinking skills, this paper advances eight hypotheses regarding the failure of research to confirm expected positive effects, and makes two major claims. The hypotheses are as follows: (1) a cognitive chain of consequences of programming instruction exists, and students are not progressing to the end of the chain; (2) applications represent a more likely area than programming for the desired cognitive outcomes; (3) research on cognitive outcomes of programming has been poorly conceptualized; (4) such research has been unsophisticated and done at the wrong age level; (5) the anticipation of cognitive benefits constitutes a resurrection of the discredited concept of mental discipline; (6) problem-solving, higher-order thinking, and other goals of programming instruction are discontinuous with the regular curriculum; (7) problem-solving and higher-order thinking may be domain-specific; and (8) failure to find the desired effects of programming has been due to a lack of curricular sophistication, and objectives related to such outcomes have not been inherent in experimental treatments. The first major claim is that the principal weakness of research on the cognitive consequences of programming instruction very likely has been its inadequate consideration of curriculum issues. The second claim is that a relatively new declarative programming language, Prolog, which is radically different from procedural languages like BASIC and Logo, merits serious consideration for educational use. A brief introduction to Prolog concludes the paper. A list of 44 references is included. (MES)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Minor changes have been made to improve reproduction quality.
Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED286463

EDUCATIONAL COMPUTING AND COGNITIVE SKILLS: CURRICULAR ISSUES AND PROLOG PROSPECTS

Roger P. Johanson

Coe College, Cedar Rapids, IA

ABSTRACT: Spearheaded by the writings of Seymour Papert, many educators have looked to computer use in education to usher in a new era in which the development of higher-order thinking skills would be promoted in the schools. Early research aimed at showing the positive effects of programming instruction on students' thinking skills was not encouraging. More recent research is only somewhat more promising. This paper begins with summary and critique of the research, advancing eight hypotheses regarding the general failure of the research to confirm the expectations. Two major claims are made. The first is that the principle weakness of research on the cognitive consequences of programming instruction very likely has been its inadequate consideration of curricular issues. The second claim is that a relatively new programming language, Prolog, which is radically different from procedural languages like BASIC and Logo, merits serious consideration for educational use. The paper concludes with a brief introduction to Prolog.

It bears noting because it is frequently overlooked: Educational computing is in its infancy. The idea of teaching machines predates adequate computer development. Instructional computing pioneers like Suppes were quick to claim the new computer technology for education. Papert's work now spans two decades. But the emergence of microcomputers, a mere ten years ago, represents the event that truly launched widespread computer use in education. Our conceptions of what the technology is and our visions of how it may be used will surely continue to evolve.

An air of impatience seems to pervade much of the literature on educational computing. Research confirmation of our expectations for computer use is too slow in coming. While we may reasonably become discontent with disk access times that exceed a few seconds, we need to be careful not to expect the time requirements for educational outcomes to match the speed of the computer. I begin with an attempt to develop some sense of perspective.

IR 013 495

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Roger P. Johanson
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Computers clearly represent a tremendously significant development in our culture. They are radically affecting business, medicine, science, the arts--virtually all aspects of our lives. It is reasonable to expect that schools would incorporate computers into their programs of instruction.

But there are forces resisting the introduction of computers into education. Among these forces is economics. Computers are expensive and many schools don't have adequate funds for such major expenditures. Furthermore, the curriculum is full. Any additions require us to rethink existing objectives and make some reductions or eliminations. Computer use, like any other instructional innovation, must be justified.

Prominent among the justifications is a claim that computer use may produce unique educational benefits. Simplistically stated, computers are logical devices, and programming is a cognitively demanding activity characteristic of the best problem solving. Papert (1980, 1984, 1987), the primary spokesperson for programming instruction, has argued convincingly that the computer represents a fertile ground for the growth of cognitive skills. Indeed, much of the enthusiasm which has accompanied the introduction of computer usage into the instructional programs of elementary and secondary schools is due to optimism about anticipated positive effects of computer use on students' thinking skills.

COMPUTER PROGRAMMING AND THINKING SKILLS

At this point, it will be helpful to delimit this discussion. To some extent, computers are like books, film strips, overhead projectors, or other forms of instructional media: they are content-neutral. They are carriers of the instruction. Computer use does not force any <u>a priori</u> curricular decisions upon the educator. The simple use of the computer to assist regular instruction is not the object of attention in this study. What is of concern is instructional use of the computer in situations where there is an explicit belief that such use will be instrumental to accomplishing objectives beyond those embodied in the content being taught. That is, the primary object of interest here is computer use when such use <u>in itself</u> carries with it curricular objectives.

An initial and widespread educational use of the computer has been for teaching programming. But the justification for this instruction, especially in many computer literacy programs, extends beyond the primary content of the specific language being taught. "The avowed purpose of most programming courses is to teach problem solving or reasoning as well as to teach programming (Linn, 1985, p.14)."

The belief that programming can have such consequences is rooted in a variety of assertions. Linn (1985) and Dalby and Linn (1986) have highlighted some of these. Programming requires decomposing complex problems into subproblems. Related generalizable skills which may be learned through programming and

which may be applicable to other problem solving tasks include use of analogy, working backwards, and top-down design. Programming becomes the training ground for practice of a variety of problem solving heuristics.

Certainly though, the belief that computers could alter intellectual development rests in large part on the work and writing of Seymour Papert(1980), especially as detailed in Mindstorms. Children learning to program a computer, Papert asserts, are thrust into thinking about their own thinking. They are turned into epistemologists. The pre-computer environment of children in all cultures is deficient in "opportunities to bring their thinking about thinking into the open (p.28)." Through Logo programming, Papert claims, children learn powerful ideas, such as debugging, which promote procedural thinking and the attainment of Piaget's formal operational thought. The destructive notion of "errors" in thought is replaced with the constructive notion of "bugs" which may be identified and corrected.

Unfortunately, results of research seeking validation of Papert's claims have been discouraging. Ehrlich, et al. (1984) concluded, that there is little empirical support for the belief that transfer will occur from programming skills to other problem solving domains. Though the activity of programming does fit definitions of higher-order thinking, the literature on whether programming leads to improvements in higher-order thinking is "thin and inconclusive" (Patterson and Smith, 1986). The results of the major ACCCEL studies indicate that while the potential for developing higher order cognitive skills through programming is seen, it has not yet been achieved (Dalbey and Linn, 1986). They conclude that "educators need to be less idealistic about teaching general problem-solving skills through programming" (p.92). While not giving up on the potential, they note that demonstrating cognitive outcomes will be difficult.

SELECTED HYPOTHESES

Why is there such discontinuity between the anticipated and observed effects of programming instruction? Can the dream and the reality be reconciled? The question has been directly and indirectly addressed by many. A review of the literature suggests a number of related hypotheses. While neither exhaustive nor entirely distinct, eight hypotheses on the failure of the research on the cognitive consequences of programming instruction to corroborate the confidence placed in such instruction follow.

Hypothesis 1: "A cognitive chain of consequences exists; students are not progressing to the end of the chain, but could." Linn (1985) and colleagues have proposed a model for understanding how programming instruction may lead to generalized cognitive growth. The model suggests the existence of a chain of cognitive consequences from programming courses. The chain begins with

learning language features, progresses to design skills (consisting of using templates and procedural skills), and, only after these have been accomplished, potentially results in generalized problem solving skills.

The research reveals a "tremendous range of student performance after an introductory course" (Linn, 1985, p.26). Details of this research can be found in Dalbey and Linn (1986). In twelve-week courses in BASIC, they found that very little programming was actually learned. While students became reasonably competent at understanding single BASIC commands, they performed poorly on measures of reformulating programs and designing complete programs. More recent research by Kurland et al. (1986) calls into question the likelihood of students ever achieving mastery of programming within our schools. Even after two years of study, many students had not learned enough to have any hope of transfer effects to other disciplines or problem areas.

While many have noted the shortcomings of BASIC as a programming language, Dalbey and Linn argue that its availability propels its use and consequently it must not be abandoned by researchers. There are important suggestions which grow out of this first hypothesis. Instruction in any single language seems inadequate; such limited exposure is unlikely to promote students' recognition of the general applicability of design skills. Augmentations to BASIC may help. The one suggested by Dalbey and Linn is Spider World, a graphics-based programming environment with limited language features. Other suggestions are addressed in the hypotheses which follow.

The research of Kurland et al. (1986) must be viewed as a significant challenge to the implications of the cognitive chain hypothesis. Even if the chain construct is accepted, it is unlikely that we can provide programs in schools which do more to move students along the chain than what was outlined in the Kurland research. Rigorous instruction in six programming languages was provided. Even given instruction "unrealistically more intensive than that found in most schools today," most students did not show evidence of the transfer which should occur at the third link in the cognitive chain. As a final note, Mandinach and Linn (1986) have reviewed the extension of the cognitive chain construct to "computer learning environments" beyond programming.

Hypothesis 2: "Applications represent a more likely arena than programming in which to look for the desired cognitive outcomes." Lockheed and Mandinach (1986) have identified as trends in instructional computing a decline in interest and a shift away from programming. The integration of applications programs (such as word processors, spreadsheets and database management systems) into the curriculum, it is argued, offers greater potential for promoting students' thinking skills.

Since applications deal with "generic functions such as

data manipulation and word processing" (p.25), they can be applied across subject areas. This would permit concentration on "generalizable skills related to planning, gathering, and interpreting data" (p.23). Part of the rationale includes an assertion that computer literacy courses dealing with programming are generally poor in quality and have a restricted curriculum. It is unrealistic to expect that these courses will be improved for most students. Applications-based instruction would be easier for teachers to provide and would allow direct access to the power of the computer to transform human reasoning capabilities. (As a note of dissent, it should be observed that Hawkins and Sheingold (1986) found that teachers using database programs had difficulty incorporating them into their curriculum.)

Lockheed and Mandinach further assert that students in general do not enjoy programming instruction. The result of this may be seen in the significant decline in recent years in the number of high school students who plan to pursue college study in computer science.[1] While no comparative studies were done, evidence is cited that students have positive attitudes regarding CAI, turtle graphics and games, but negative attitudes toward BASIC programming instruction. However, Johanson (1985) found evidence of students' enjoyment of computer literacy instruction centering on BASIC. Similar empirical and anecdotal evidence is not hard to find.

Research concerning the use of applications programs in education is just beginning. Nonetheless, Lockheed and Mandinach claim that results using a curriculum based on applications software are likely to include both acquisition of higher cognitive skills and greater positive student response to the computer. (While affective outcomes are not the concern of this study, it should be noted that Ginther and Williamson (1985) claimed that there is much more support for personal-social benefits from Logo instruction than for logical-analytic outcomes.)

Hypothesis 3: "The research on cognitive outcomes of programming has been poorly conceptualized." Pea (1984) noted the disturbing failure of research to validate the claim that Logo instruction would yield cognitive dividends. Perhaps, he reasoned, the research has failed because it was inappropriately focused. Maybe <u>general</u> intellectual benefits are not the outcomes to seek. Concrete, functional perspectives may be required. The culture of the Logo classroom has not been sufficiently considered. "Without some functional significance to the activities for those who are learning the new practices [associated with Logo], there is unlikely to be successful, transferable learning" (p.8). Intellectual benefits should only be anticipated within the context of a culture in which Logo serves a meaningful purpose for the members of that culture.

In a recent response to the problem of research verification of the significance of Logo, Papert (1987) has criticized the

work of Pea and others at Bank Street. Papert notes that to expect to hold most of the educational experience constant while testing for the effects of Logo is part of a mistaken research paradigm which cannot be adequate to the task. Within the context of a Logo instructional experience as intended by Papert, students "don't encounter a thing, they encounter a culture" (p.27). By analogy to literacy research, Pea (1984) has acknowledged that locating Logo instruction within a supportive culture is essential to assessment of the Logo instructional "treatment". Yet the differences which would then exist between the instructional treatment and the control would raise our collective methodological eyebrows. Papert argues that there is a "radical incompatibility" between the study of Logo and the traditional educational research "treatment" method. He suggests that we can view the research designs which prevent us from establishing and evaluating significant educational alternatives as being suspect rather than the alternatives themselves.

A rather different problem is also best located within the context of this third hypothesis. Agreement upon definitions and choices of anticipated outcomes of computer instruction has not been reached. While the term "cognitive consequences" is appealing, it masks the variety of outcomes that have been studied. Among these are cognition or cognitive development, problem solving ability, higher-order thinking, thinking skills, and metacognition. As Pea's analysis suggests, it may be time to concentrate on identifying more specific, concrete outcomes. Patterson and Smith (1986) note that there is no agreement on what constitutes higher-order thinking. By identifying less grand and glorious targets than those listed above, we may be more likely to find encouragement in the research efforts. Note however that this tends to return us to the research paradigm rejected by Papert.

Hypothesis 4: "Research has been unsophisticated and done at the wrong age level." Specific and measurable aspects of cognitive functioning must be identified and studied. Furthermore, since the age of about six or seven is the time generally acknowledged as the transition period from Piaget's pre-operational to concrete operational stages, this is an age which may be ripe for identifying developmental differences. Papert (1980) asserted that Logo would be influential in children's cognitive development for two primary reasons. First it provides the raw material (which is deficient in pre-computer cultures) for the development of operational thought. Second, children would be transformed by Logo activities into epistemologists, thinking about their own thinking. The first requires a depth and duration of exposure which probably has not been achieved in empirical studies to date. By working with younger children the importance of the treatment relative to the child's other experiences is increased. (A six month treatment constitutes 8% of a six-year-old's life but only 4% of a twelve-year-old's.) The second reason requires researchers to focus on measures of

metacognitive activity. If an increase in children's metacognitive abilities can be shown, then the hypothesis that the eventual result will be an improvement in cognitive functioning is more plausible.

Among the most promising research in the field is that of Clements and colleagues. In a recent study, Clements (1986) examined the effects of Logo and CAI on several components of students' cognitive and creative capabilities. Perhaps critical to his findings was his decision to use first and third grade subjects. Also important is a recognition that it is not simply Logo, but a Logo "environment" which is responsible for the effects observed. When compared to the CAI and control groups, Clements found that Logo had positive effects on classification and seriation (for first graders but not third graders), five of six metacomponents (four related to deciding on the problem and solution processes, and comprehension monitoring), the originality and elaboration subscales of the Torrance creativity measures used, and a measure of children's abilities to give directions.

It may be argued within the confines of this hypothesis that the research on programming is just now becoming sufficiently sophisticated to discover the cognitive effects. Extended treatments with younger subjects, and identification and measurement of specific anticipated outcomes such as aspects of metacognition and components of creativity could pave the way for more encouraging results. Bransford, et al. (1986) assert that inattention to metacognition is the critical error of previous research in this field.

Hypothesis 5: "The anticipation of cognitive benefits constitutes a resurrection of the discredited concept of mental discipline". Despite failures to document the claim that instruction in Latin, geometry and other disciplines has discernable effects on the development of thinking skills, many educators apparently believed that computers, with their seemingly limitless potential, would be different. There is an undeniable similarity here to the once popular concept of mental discipline. The seductive notion that certain instructional activities could serve as a type of mental exercise which would strengthen the intellect was widely held but also widely discredited in the early part of this century. The works of Thorndike and Woodworth (1901) and Judd (1908) are representative. The conclusion reached was that general intelligence is not affected by training in specific domains. There is within the more enthusiastic rationalizations for Logo a rather haunting echo of those claims which early psychologists rejected.

What survived from the early research was the less sweeping notion of transfer. The generally accepted principle is that transfer occurs between the learning of two intellectual skills in proportion to the degree of similarity between the two skills. Thus we might anticipate that learning one programming language will transfer to learning another. Planning skills learned in

programming might transfer to other planning tasks. But general cognitive gains should not be anticipated. Hoffman (1985) argues, "Cognitive development is extremely complex and psychologists as well as commercial software developers are not able to facilitate its growth" (p.360). Even transfer is by no means assured. Ginther and Williamson (1985, p.76) conclude, "Decades of research on problem solving, with both animal and human subjects, has shown that transfer of general problem-solving skills is difficult to achieve." The logical conclusion of this hypothesis would have to be that expectations of significant cognitive benefits from programming instruction are naive.

Hypothesis 6: "Problem solving, higher-order thinking, divergent thinking and other goals of programming instruction are discontinuous with the regular curriculum and are unlikely to be achieved." Papert sets the tone in the introduction to Mindstorms when he states that what he proposes goes in the opposite direction of what schools are doing. Walker (1986) observes that current educational practice has a strong molding influence on how computers are used. One must ask whether this is not a case of irresistible force and immovable object. Indeed, it appears that schools are more immovable than the Logo philosophy is irresistible. Hawkins and Sheingold (1986) observed that teachers and students were uncomfortable with the ambiguousness of Logo. They furthermore claim that there is an incompatibility between the traditional curriculum and problem solving or critical inquiry emphases. Patterson and Smith (1986) reach the conclusion that schools tend not to encourage higher-order thinking.

Debates on curricular issues are certainly not new. Computer programming instruction simply represents a new field of battle for old views concerning what knowledge is of most value and how best to teach it. Walker (1986) notes some parallels. "The debates about BASIC vs. Logo . . . could, with only light editing, be applied to content coverage vs. problem solving skills" (p.27). He does, however, suggest that change may be inevitable--that we will not continue to be satisfied with stereotyped talking and passive reading in schools. The current tension which exists between standard school practice and what Logo advocates are trying to accomplish (including cognitive development, problem solving skills and reflective inquiry) may account for much of the difficulty to demonstrate significant effects of programming instruction. It is virtually impossible to eliminate or even account for the effects of the traditional curriculum on students' perception of the task at hand in learning to program. Within the context of traditional school practices, most students may simply be unable to make the mental shift necessary to allow the cognitive skill goals of programming instruction to be fostered.

Hypothesis 7: "Problem-solving and higher-order thinking may be domain-specific." This claim has been noted in Patterson and Smith's (1986) review. Linn (1985, p.15) notes that "cognitive scientists have shown that learning is much more discipline-specific than had been thought." Students who learn to solve problems in the context of programming, she observes, are not necessarily learning to solve problems in other contexts. She cites the research of Chi, et al. (1981) which concludes that the structure and organization of knowledge are critical to its subsequent use by learners. But it has not been claimed that the acquisition of programming competence has any relation to such issues as knowledge structure. Nor does it seem reasonable to claim that programming in BASIC or Pascal or even Logo will have any impact on the programmer's cognitive structure except for that portion of it directly related to programming concepts. The conclusion must be that programming instruction can have only very limited impact on problem solving and higher-order thinking.

The work of Novak (1977) based on Ausubelian theories of learning gives theoretical foundation to the claim that the organization of appropriate (domain-specific) cognitive structure is the primary factor which accounts for problem solving ability. There has been a significant trend in recent years for instructional programs designed to teach thinking and problem solving to emphasize the importance of domain-specific knowledge (Bransford, et al., 1986). The degree to which a commitment to planning, problem decomposition and sustained inquiry, which may be gained from computer programming, can alter problem solving skills is thus relatively minor, and it marks the extent of the potential of programming instruction to improve problem solving in non-programming domains. True improvement in higher-order thinking ability depends on enhancing cognitive structure in the domain in which that ability is to be demonstrated. Programming instruction must then be of little consequence.

Hypothesis 8: "Failure to find the desired effects of programming, such as higher-order thinking, problem solving, and enhanced metacognition, have been due to a lack of curricular sophistication. Objectives related to such outcomes have not been adequately inherent in the experimental treatments."

The partial answer for the failure to obtain evidence of transfer from computer use to thinking skills advanced here is based on a simple curricular notion: students learn what they are taught. Walker and Schaffarzick's (1974) important review of decades of educational research, "Comparing Curricula", notes that when differences in the outcomes of two instructional programs compared were found, they were typically due to differing curricular intentions.

When we fail to find intended outcomes, we must examine the possibility that our instruction does not embody our objectives. Instruction in procedural programming languages like BASIC, Pascal and Logo is demanding. Mastery of the syntax and programming principles dominates instructional time and thus

effectively dominates the curriculum. As previously noted, Linn's (1985) hypothesized chain of cognitive accomplishments begins with learning language features and ends with learning problem solving skills. Since instructional time is limited, it is quite possible that progression to the end of the chain--where the curricular intentions reside--will not occur. Dalbey and Linn's (1986) research suggests that language features tend to dominate junior high school programming courses. They found little evidence that students learning BASIC engaged in planning activities.

Indeed this curricular hypothesis appears the strongest one. Several of the previously cited authors have directly or indirectly appealed to this hypothesis and we return to their comments now. Pea and Kurland (1984) and Kurland, et al. (1984) noted that planning is not a necessary outcome of Logo instruction; it must be explicitly taught. (Somewhat more pessimistically, they note that achievement of expert programmer status takes enormous amounts of time--perhaps 1000 hours. The best experimental treatments reported in early research involve no more than about 100 hours.) Similarly, Linn (1985) observed that students' self-generated design templates may be an impediment to developing problem solving skills. It appears that in order for design skills to be generalizable, they must have been taught. She concludes that considerable curricular improvement will be necessary for problem solving outcomes to occur. Dalbey and Linn (1986) found that teachers' expectations seem to be limited to design features and students' assignments tended to be drills on language features.

The criticism cannot be limited to instruction in BASIC. In his analogy to research on the effects of literacy, Pea (1984) asserted that it is a mistake to think of powerful ideas as being inherent in Logo itself. While Logo enables recursion, structured procedural programming, debugging, modularization and documentation, it is unreasonable to expect that many students will spontaneously "discover" such tools. In fact the evidence is strong that most students do not. Furthermore, when the best of instruction within a Logo culture occurs, it clearly involves a teacher who promotes use of the tools just listed as well as children's epistemological activity. "To call these [supportive, stimulative, instructional] activities 'learning without curriculum' is misleading, and an overly narrow view of what constitutes curriculum." (p.7)

Let us return now to the work of Clements (1986). The evidence he provides of the desired and anticipated positive effects of Logo instruction on cognition is encouraging. However it must be argued that it is the curricular intentions embodied in the Logo instructional procedure used in that study which account for the effects found. Careful attention to the treatment discussion reveals key components which are not dictated by the Logo language. Children were instructed to begin by drawing the picture they wanted the turtle to produce. Then they decomposed their drawings into discrete components which

were individually traced and which became the models for procedures to be programmed. (It is not explained how much guidance was given at this point.) Two support programs which enhanced feedback during programming were employed. This allowed students to edit programs at the same time as viewing the effects of individual commands invoked. "If a procedure was not doing what the children had anticipated, they were encouraged to think it through," by being asked appropriate debugging questions (emphasis added). A carefully designed sequence of 44 instructional sessions was used. While Clements indicates that didactic presentations were not employed, procedural thinking and debugging were explicitly taught. He concludes by noting that adult guidance was given frequently. The research design provided that one or two instructors were present during each of the sessions which were taught to six students working on three computers.

It is in no way argued here that such instructional procedures are inappropriate. Indeed they are commendable. I simply claim that it is the explicit curricular intentions concerning metacognition and refinement of students' thinking which are the likely causes of the demonstrated effectiveness of the Logo instruction. It remains a reasonable claim that Logo is a uniquely powerful vehicle for permitting and even encouraging teacher behavior related to cognitive curricular objectives. Indeed one control group in the Clements study received similar (though much less extensive) guidance as they worked on CAI lessons, yet the Logo group's post- treatment performance on several key assessment variables was superior. As has been noted, the regular school curriculum gives very little attention to promoting higher-order thinking. Logo may well be one of the most appropriate environments in which to introduce instruction aimed at enhancing student cognition.[2] But no research on the effectiveness of Logo to produce cognitive growth is likely to be successful unless it includes an instructional treatment which ensures that cognitive growth is represented in the curricular objectives.

RESEARCH: SUMMARY COMMENTS

In a broad sense, what is at issue here is the ever-present question of the appropriate aims of schooling.How much do we agree that teaching students "to think" is a primary goal of our educational programs? In questioning the very nature of the research reviewed here, Papert (1987) returns this question to the forefront of the debate. "We may have to reexamine assumptions about education that were made long before the advent of computers. One could even argue that the principal contribution to education made thus far by the computer has been to force us to think through issues that themselves have nothing to do with computers (p.23)." Papert urges us to consider the people and the cultures through which Logo becomes known by

students if we hope to truly study the effects of this new technology.

The "research" most needed now is curricular and instructional development. We need not be dependent on technocentric thinking or narrow research paradigms. Let us instead seek to develop complex school cultures in which computers play significant roles in helping students and teachers to engage in higher-order thinking. Let us examine these to see what aspects may deserve controlled research attention. The design of curriculum and instruction must not await research verification of effectiveness. (Indeed it frequently does not.) Unfortunately we seem to be at a point where negative research results may discourage development efforts in instructional computing. We need to trust our professional judgments and intuition at least as much as our research methods.

As a preface to the section which follows, let us compare the concerns about Logo to the comments of Smith (1986, p.110) with respect to instructional efforts using Prolog. "We are in a truly unusual situation where classroom developments in Information Technology applications are far ahead of the attempts of either bureaucrats to regulate them or of academics to understand them. We are in the middle of a practitioner-led revolution!" Ennals and Nichol "have inspired a small but important community of 'barefoot curriculum developers'."

This is not a call to cease the kind of research efforts reviewed earlier. However, it is an attempt to examine weaknesses of that research. Two responses are appropriate. First, we can try to improve the research. The hypotheses advanced here suggest some directions for that. Second, we must continue to try to use computers in ways that we have reason to believe may have "cognitive consequences". There is good reason to be undeterred in those efforts. A combination of research and anecdotal evidence supports continued use of Logo. The second part of the present paper suggests another potentially fruitful direction for instructional computing using Prolog.

Perhaps it will not or cannot be shown by controlled research that acquisition of programming competence has important cognitive effects. That does not preclude rational argument that we should teach programming and other computer-based activities with an expectation that a long term consequence of such instruction will be the development of higher-order thinking and/or problem solving. Nor does it in any way argue against these as legitimate curricular concerns. Assuming that we indeed wish to promote them, failure to demonstrate cognitive consequences does not argue against such use of the computer <u>unless</u> an alternative instructional approach has been verified as effective, or can more logically be argued to be effective. It simply says that we will have to <u>continue</u> to hold curricular analysis as the best decision tool for selecting among instructional strategies.

AN ALTERNATIVE LANGUAGE - PROLOG

A good deal of Papert's (1980) effort has gone into arguing that traditional programming languages are inadequate to the task of stimulating cognitive development. Under Papert's leadership, Logo has achieved the status of a mainstream educational programming language. Similarly, arguments that some language (other than BASIC) should be adopted for educational use have been advanced for a wide variety of languages including Pascal, PILOT, Smalltalk, and COMAL. While languages are certainly not a dime-a-dozen, their proliferation is staggering. The journal, Computer Language, includes as a regular feature their "Exotic language of the month club". It is not without considerable risk that one makes the claim that another new language deserves consideration. However, that is precisely what I shall proceed to do.

Wittgenstein (1961, p.115, proposition 5.6) asserted that "the limits of my language mean the limits of my world," and his claim rings true for computer languages as well as for "natural" ones. As the needs and purposes of programmers and other computer users evolve, it becomes clear that there are language deficiencies in many programming languages which inhibit or even prohibit certain functions. New languages are developed in large part to extend the limits imposed by old languages. The language advanced here is Prolog, for programming in logic. There is a radical difference between Prolog and the other languages mentioned here. Logo, BASIC, Pascal and most commonly used languages are procedural. That is to say, programming in such a language consists of creating procedures which the computer will execute in order to solve the particular problem at hand.[3]

Unlike any of these other languages, Prolog is declarative. As such, programming in it involves describing facts and relationships about a problem rather than outlining a procedure for the computer to follow in solving it. The user must focus attention on declaring or clarifying the problem but can leave the computational details to the computer and the "inference engine" which is built into the language. Prolog is an excellent representative of important trends in computer science such as parallel processing (while not necessary for Prolog, it is easily facilitated), non- determinism, and pattern-directed procedures, as well as declarative rather than procedural approaches. Originally developed as a language in which mathematical proofs might be achieved mechanically, Prolog has gained prominence in artificial intelligence and now competes favorably with LISP. Its recent selection for use as the language of the Japanese Fifth Generation computer project has significantly boosted interest in its study. While it is unfamiliar to most people in the educational computing community of the U.S., it may offer significant opportunities for accomplishing elusive curricular goals.

Considerable use of PROLOG in school settings has begun in England. Much of this work involves a special use of PROLOG not

easily matched by other languages. In it, a "database" of information is established, which can then be "queried." The emphasis then becomes disciplined inquiry into any body of curricular content (Nichol, et al. 1985). One of the leaders in the field, Richard Ennals (1984) suggests, "the computer should not be the focus of attention. We should return to thinking about education and training, and the thinking that we wish to encourage, with the computer among our tools."

It is not the intention to provide a Prolog tutorial here. The standard reference on Prolog is Clocksin and Mellish (1981). More readable introductions for those not wishing as rigorous and thorough a treatment include Rogers (1986), Bharath (1986), Cuadrado and Cuadrado (1985) and Bratko (1986). As with many languages, there are a variety of dialects of Prolog. Prominent among these is micro- Prolog. A good introduction to micro-Prolog and educational uses of Prolog in general can be found in Ennals (1984). While these references should be consulted for more complete discussion of the language, a brief overview of Prolog will aid the present discussion.

Programming in Prolog includes three primary activities: asserting facts, establishing rules, and querying the database of facts and rules. Examples of each follow. Note the similarity to natural language; the meaning of many of the statements should be obvious. Comments are enclosed, as they are in Prolog, between the symbols /* and */.

```
/* facts:                */
four_sided(rectangle).       /* a rectangle is four-sided */
animal(mouse).               /* a mouse is an animal      */
eats(owl,mouse).             /* an owl eats a mouse       */
ended(world_war_II,1945).
book(mindstorms,papert,1980,basic_books).

/* rules:
   The general form is:      CONCLUSION  IF  CONDITIONS;
   the symbol   :-   means "if" or "when";
   variables begin with uppercase letters.    */

quadrilateral(Figure)  :-  four_sided(Figure).
 /* A Figure is a quadrilateral if that Figure is four-sided.  */

carnivore(Animal1)  :-  eats(Animal1,X), animal(X).

/* This second rule says essentially "Animal1 is a carnivore
if there is something that it eats and that something is an
animal."  X has been used as a variable here because it is
probably more easily comprehended by novices.  The comma
between clauses has the meaning "and".   */
```

```
/* queries:                    */
ended(world_war_II,When).        /* The computer will
                                    respond:
                                    When = 1945  */
four_sided(rectangle).           /* response:
                                    Yes          */
carnivore(What).                 /* response:
                                    What = owl   */
book(mindstorms,Author,_,_).     /* This simply asks who the
                                    author of mindstorms is.
                                    response:
                                    Author = papert  */
four_sided(square).              /* response:
                                    No           */
```

Note that for the last query, the answer "no" must be interpreted as meaning that it cannot be verified from the database that a square is four-sided.

The combination of the facts and rules given above constitute a "complete" Prolog program which will answer the questions listed in the query section. The critical observation is that the procedure which the computer should use to answer the questions listed is not spelled out in the program. It is resident in the language and largely transparent to the user. (The procedure consists largely of pattern-matching, depth-first search, and backtracking.) The "programming" consists of declaring the facts and rules and then asking appropriate questions.

This crude introduction will have to suffice. Following a brief overview of present and potential educational uses of Prolog, the remainder of the discussion addresses some of the less obvious consequences of using Prolog and forms a beginning of the rationale for its consideration for educational purposes.

INSTRUCTIONAL USES OF PROLOG

Only a brief introduction is included here. For further details, good sources include Yazdani (1984), Ennals (1984), and PEGBOARD, the newsletter of the Prolog Education Group at the School of Education, University of Exeter, England.

The most obvious use of any programming language involves programming instruction. When children (or adults) learn to program in Prolog, they must become involved in logical analysis and specification of knowledge. Relationships between objects, events and concepts must be specified. As previously noted, much less attention is given to procedural issues. However typical input-output and numerical manipulation problems must generally be avoided at first. A prototypical problem in Prolog would be to lay out a family tree by identifying parent-child relationships and the sex of individuals listed, and then to

specify rules to allow answering questions like who is the cousin of whom, who are the descendants of Martha, and does Sam have a known great-grandmother. Logic is the basis of Prolog programming, but very little understanding of logic is necessary for getting started in Prolog. Indeed since much of the procedural detail is hidden within the language itself, the demands on the novice programmer are relatively small.

Those working with educational applications of Prolog in England have concentrated on two other uses of the language--1) working with already developed Prolog database programs, and 2) utilizing various front-end programs, authoring toolkits or expert system shells written in Prolog. The database approach is well represented by an "classic" program in this category, BOGBOD (Nichol and Dean, 1984). Students are provided a problem to solve concerning a body which has been found in a peat bog. A variety of information concerning this problem is represented in the database. Students work collectively querying the database trying to form their own hypotheses on who the dead person was, who killed him and what the circumstances were. This problem is a representation of an actual historical event. The program allows students to engage in historical inquiry rather than to simply commit facts to memory. This is in line with what some see as a highly significant trend in school practice--a movement toward promoting active student involvement in learning. (For example, NEA's 1987 spring conference featured this topic, bearing the title, "Revolution in Learning: The Student as Active Learner".)

An example of the toolkit use of Prolog, Linx, bears some similarity to an expert system shell approach. It is intended as an aid to writing simulations. The Linx manual (Briggs, et al., in development, p.2) states that with the toolkit, "we have written simulations about sea voyages and battles, classification trees and taxonomies for the study of rocks, early man and insects and language tools for the teaching of English." Similar to authoring languages, such uses of Prolog allow teachers (and students) to write their own instructional programs, building databases which will then be available for query. A significant feature of Prolog which supports this type of use is that there is virtually no distinction between data and program. Programs can be self- modifying. The importance of this category of Prolog use is noted by Brough (1986, p.28). "The concept of developing open or extensible Prolog-based shells to support teachers across the curriculum is a very exciting one and is particularly relevant to the 'new curriculum' being developed to free subjects from the older facts-based approaches." Unlike older authoring languages, Prolog-based tools are not oriented toward fancy screen displays and programmed instruction approaches. What is most directly supported is development of programs which encourage disciplined inquiry into a body of content.[4]

While requiring some programming emphasis, the last category of uses of Prolog to be discussed here includes a variety of

activities intended to focus on conceptual development. Recall our previous Prolog examples:

```
four-sided(rectangle).          /* a fact  */
quadrilateral(Figure)  :-  four-sided(Figure).
                                /* a rule  */
```

When we inquired whether a square was four-sided, the response was "no". Clearly the database needs to be amended to correct this omission of information. There are numerous ways to do this though, and in the process, students can struggle with issues of necessary and sufficient information. For example, we can ask, "given the database, how can we get correct answers to the questions 'is a square a quadrilateral?' and 'is a square four-sided?' by adding the least information to the database?" Of course one way would be to simply add as facts:

```
quadrilateral(square).
four-sided(square).
```

Students can try various approaches, while utilizing and clarifying their concepts, eventually arriving at a smaller representation of the information such as:

```
four-sided(square).
```

The fact that a square is a quadrilateral is now inferred by Prolog from the rule which says any four-sided figure is a quadrilateral. This very simple example should demonstrate the idea while suggesting similar efforts in any discipline.

A slight variation is to present a database which contains redundant information. Students are now given a set of queries which must continue to be answered correctly, but are asked what information may be removed from the database. Other similar uses involve groups of students creating their own databases of information for sharing with other groups. They then engage in researching and storing information on an assigned or self-selected topic. When finished, they identify sample queries for another group of students to use in accessing the information embodied in their database. The prime benefit of course is in the creation of the database, but the sharing provides an audience to give purpose to their efforts.

This list is suggestive rather than exhaustive. Whether instructional activities are done directly in Prolog, or with the aid of Prolog-based tools, it is easy to see that support for problem solving and conceptual development is central. Nichol, et al. (1986, p.89) note, "A pupil who writes a program using an IKBS [intelligent knowledge based system] or expert systems shell learns from the researching and structuring of the knowledge which the program represents." As a final comment on Prolog uses,

the power of Prolog as a software development tool should be explicitly noted. "There are major benefits in the declarative as opposed to the procedural approach to software specification and development" (Brough, 1986, p.20). Both the professional developer and the classroom teacher may benefit from use of Prolog to try out instructional ideas.

CONSEQUENCES OF AND RATIONALE FOR USING PROLOG

Prolog requires--or allows--a radical reorientation to programming. Procedural considerations become secondary. The problem being solved becomes primary. Knowledge organization, consideration of necessary and sufficient information, conceptual relationships and logical analysis become appropriate considerations when programming--and these all arise rather naturally. At least for beginning applications, the programming language comes close to disappearing. No longer is it necessary to resort to mathematical problems involving counting loops or iterations or distinguishing reals from integers or turtle geometry. It is not my intention to belittle such activities; they represent important concepts from math and computer science. Computers are powerful tools for mathematical concept development, but other disciplines have been slighted by instructional programming. With Prolog, the content of any discipline can be naturally transformed into a computer program.

Let us return to our original concern, whether computer programming facilitates cognitive development, problem solving and acquisition of general thinking skills. Much of the claim that it does (or might) is based on arguments involving the use of procedural thinking. If programming in Prolog doesn't depend primarily on procedures, how could it have important cognitive consequences? If our primary curricular objectives are related to thinking skills, why should Prolog be considered? The remaining discussion will be limited to these questions.

As noted earlier, Linn's work suggests that there is a chain of cognitive consequences to be expected from learning programming. Before generalizable problem solving skills are acquired, the learner must progress through the stages of learning language features and design skills. Her research indicates that few students in computer literacy classes actually do progress through these first two stages. As a result, it is unlikely that students learning to program will achieve the desired terminal problem solving objectives. However, Prolog's language features are minimal. It should therefore be a simpler matter with Prolog to progress through the chain.

The issue of the dichotomy between content knowledge and cognition implied when we talk about thinking as a skill has been neglected so far in this discussion. Nickerson, Perkins and Smith (1985) challenge the popular notion that thinking skills and knowledge are distinct, suggesting rather that the two are at least interdependent. Education must address both thinking skill and knowledge objectives, they argue. Part of the failure for

programming to demonstrate improvements in thinking skills may be due to its having largely failed to make significant attachments to bodies of content. This is a potentially crippling weakness. Recall the observations, based on reviews of the literature in the field, that problem solving may be domain dependent (Patterson and Smith, 1986; Linn, 1985; Bransford, et al., 1986).

The specification of relationships between concepts is an integral component of Prolog programming. Indeed it is impossible to program in Prolog without addressing some body of content. And it is the content, not the programming language, which captures the largest share of the programmer's attention. This has considerable significance for the issue of developing problem solving and thinking skills. Understanding relationships among concepts and procedures is characteristic of expert problem solvers (Patterson and Smith, 1986). Knowledge organization and structure play a major role in how that information is later used (Chi, et al., 1981). Prolog forces the user to clearly declare the relationships which exist among concepts under study. It is reasonable to assume that the level of specificity required by the computer will promote the user's own understanding of the knowledge represented. Prolog provides a "catalyst for thinking about the subject of interest. The program that may result at the end of a lesson is not necessarily the primary objective, but a by-product of a thinking activity" (Ennals and Briggs, 1985).

Another argument in favor of Prolog follows from the suggestions of those calling for applications-based computer instruction. An applications emphasis would center on "generalizable skills related to planning, gathering, and interpreting data" (Lockheed and Mandinach, 1986, p.23). But this is precisely the focus of Prolog. The language has already gained prominence as an excellent tool for use with relational databases (Moss, 1987). Hawkins and Sheingold (1986) noted that teachers using database programs had difficulty incorporating them into their curriculum. Some of this was due to the business orientation of programs not written for educational use. Complete control over how the information in a database is organized is provided by Prolog, thus potentially alleviating much of this problem.

New technologies appearing and soon to appear increase the importance of information retrieval. The skills required by these technologies will require the ability to find and organize information for inclusion in a database and to extract that information once it has been stored. Prolog programming models this knowledge representation procedure and provides an opportunity to practice these storage and retrieval skills.

CONCLUSION

In the end it is curricular analysis which is likely to prove our best guide to assessing whether any instruction, including programming or other computer-based activities, can be expected to foster development of thinking skills. Since

students tend to learn what we teach them, we need to consider what we are really teaching and what our intentions are for this learning. We must acknowledge that much of schooling at present relies on low level thinking. Programming instruction may be characterized as higher-order thinking, but a great deal of the energy of learning to program goes into learning the syntax and structure of most programming languages and hence may interfere with the intended goal of problem solving skill development. Added to this is the large problem of transfer. Real-world problems exist within some domain of knowledge. Many of the problems posed in traditional programming languages for beginners are artificial and contrived and make no reference to the domains in which we will want students to solve problems.

We would do well to explore the potential Prolog may have for addressing these concerns. Teams led by Ennals and Nichol in England have been working with educational applications of Prolog for most of this decade. The work they have pioneered needs to be pursued in this country as well. Because of its limited language features, simple syntax, declarative style and clear content integration, Prolog may allow us to move closer to the goal of promoting students' cognitive growth.

I conclude with a few perhaps heretical observations. There remains little dispute to the claim that computers constitute the basis of a "revolution" analogous to those which resulted from the printing press and industrialization. Computers have changed and will continue to change our world. Furthermore, while generally well accepted, computers still engender some fear. This fear and a more general resistance to change have in large part moved us into a somewhat defensive posture with respect to instructional uses of computers. In order to justify buying computers for schools and taking time away from other instruction, we have attempted to make a claim that there is something uniquely different about computers. In fact, there probably is, but it is not what we seem to think it is. Computer based instruction is unique in its flexibility, in it's potential to accommodate an enormous range of instructional strategies. We are only beginning to explore the potential.

Prominent among the rationalizations for computer use has been the claim that computers could aid in children's cognitive development. We are now finding evidence which calls into question the hopes that there might be significant transfer from programming (or use of applications or computer games) to generalized problem solving or other higher-order thinking. Will the response be to give up using computers in schools?

Let us consider mathematics for a moment. As noted earlier, similar claims have been made for mathematics, particularly around the beginning of this century. As with programming, the claims were largely refuted. Yet instruction in mathematics continues, because it is clear that the content itself merits inclusion in the school curriculum. I think computer use and even computer programming can be justified without recourse to the cognitive consequences hypothesis. We might do well to

devote more effort there.

Finally, I end with some speculation regarding cognitive benefits of computer instruction, because I don't think we should reject this claim. Following the curricular maxim referred to above (students learn what we teach them) and insights from psychology, a fair generalization is to say that children learn to think by engaging in thinking. Indeed that is precisely the core of the dream which Papert outlines for us. Logo is to help create an environment in which children can explore their own thinking. Instructional guidance is necessary, but so is intellectual tinkering, self-directed exploration, messing around. There is very little evidence of that in the treatments which have been used in the studies which have failed to demonstrate that programming has cognitive effects. This is not surprising though, since such absence of instructional control would be methodologically unacceptable and, by conventional educational wisdom, inefficient. Herein is the ultimate dilemma. The instructional control deemed necessary may be incompatible with fostering independent, higher-order thinking.

Perhaps the cognitive benefits of learning to program are not short term. Maybe they will surface unexpectedly at times far removed from the instruction when a relationship is vaguely sensed. Perhaps the benefits are idiosyncratic. How pompous it is to think that we can pose for every student the situations in which transfer will occur so that we can demonstrate that it does. If we value higher-order thinking, we can justify computer activities which simply engage students in this kind of thinking that we too seldom incorporate into our schools' curricula.

At the heart of research design is control. We need to be abe to control enough to be confident about the effects of the uncontrolled variables. There is a fitting irony to this. For control is a central concept in schools also. We clearly believe that we should control what students learn and when. Ever more sophisticated instruction in computer programming, directed at teacher-defined objectives, is likely to be seen as merely another academic hoop through which students must jump. Why <u>should</u> this yield any different outcomes than any other school instruction?

At issue here is not simply the computer revolution and how (not whether) it finds a place in education, but also a revolution in education itself. The revolution is toward active, meaningful participation in learning by the learner. Higher-order thinking is necessarily somewhat original to the thinker--if it is simply replicative it isn't higher-order. A fair test of the cognitive consequence hypothesis would involve establishing a teaching-learning environment where students would engage in <u>guided</u> exploration of powerful ideas using a computer, with significant self-direction. The consequences would have to be looked for over a period as long as a lifetime. Even if such research could be done, it would not be the kind which would likely be published, or be chosen to guide practice. Yet there is evidence of movement in the direction of reconceptualizing

students as active learners. The computer can be a valuable tool for this. Let us continue with both revolutions--computers and active learners in our schools.

REFERENCE NOTES:

1. Such a conclusion is speculative. A much stronger claim can be made that the huge increase of earlier years was due to a societal mystique concerning computers. It is unrealistic to expect that 10% of students (the high point reached to which current percentages are compared) would become computer science majors. Students' increasing sophistication regarding computer programming resulted in more realistic (i.e. smaller) numbers of students electing to study computer science. Rather than blame computer literacy courses for turning students off to careers in computer science we should credit them with helping students to make more informed choices of college majors.

2. Actually, impassioned arguments for various curricular movements within school subject areas bear striking similarity to rationales for Logo. Papert emphasizes encouraging students' acceptance and use of bugs in their thinking. In the same spirit, arguing for a "whole language" approach to language arts, Goodman (1986) contends that children's errors are critically important indicators of their growing literacy. He urges helping learners to value "errors" such as reading miscues and invented spellings and punctuation. Much of the literature of mathematics and science education stresses problem decomposition and use of problem solving heuristics. (As noted earlier, mathematics largely lost its claim that it could develop thinking skills early in this century.) Social studies is valued for its promotion of critical thinking. Supporters of each discipline have claimed that their instruction ultimately affects cognitive development.

3. There are well known differences in the types of problems best handled by different languages. For example, FORTRAN is a good "formula" handling/number crunching language, COBOL is "business oriented", and BASIC purports to be "all purpose". Yet programming in any of these languages involves writing procedures (or algorithms) which do the work of the program. The issue here is not the category of problems for which the language is intended, but rather the way in which programming problems are solved.

4. It should be noted that some versions of Prolog give greater attention to design features. Many, like A.D.A.'s public domain and specialized versions of Prolog, include screen graphics features. Turtle graphics has been implemented in Prolog (Ball, 1984). Borland's "Turbo-Prolog" includes features for easy use of windows.

REFERENCES

Ball, D. (1984). Using PROLOG with graphics in the classroom. Computer Education, (46) February, 5-6.

Bharath, R. (1986). An introduction to Prolog. Blue Ridge Summit, PA: Tab Books.

Bransford, J., Sherwood, R., Vye, N. & Rieser, J. (1986). Teaching thinking and problem solving. American Psychologist, 41, 1078-1089.

Bratko, I. (1986). Prolog programming for artificial intelligence. Wokingham, England: Addison-Wesley.

Briggs, J., Dean, J. & Nichol, J. (in development). Linx. Exeter, England: PEG-Exeter.

Brough, D. (1986). Applications of Prolog research ideas in education. PEGBOARD, 2(1), 20-28.

Chi, M.R.H., Feltovich, P.J. & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. Cognitive Science, 5, 121-152.

Clements, Douglas H. (1986). Effects of Logo and CAI environments on cognition and creativity. Journal of Educational Psychology, 78(4), 309-318.

Clocksin, W.F. & Mellish, C.S. (1981). Programming in Prolog. New York: Springer-Verlag.

Cuadrado, C.Y. & Cuadrado, J.L. (1985). Prolog goes to work. Byte, 10(8), 151-158.

Dalbey, J. & Linn, M.C. (1986). Cognitive consequences of programming: augmentations to basic instruction. Journal of Educational Computing Research, 2(1), 75-93.

Ehrlich, K., Abbott, V., Salter, W. & Soloway, E. (1984). Issues and problems in studying transfer effects of programming. Paper presented at the annual meeting of the American Educational Research Association, New Orleans. (ERIC Document Reproduction Service, ED-257-441).

Ennals, R. & Briggs, J. (1985). Fifth generation computing: introducing micro-prolog into the classroom. Journal of Educational Computing Research, 1(1), 97-111.

Ennals, R. (1984). Future needs. The Times Educational Supplement - Extra. May 25, 43.

Ginther, D.W. & Williamson, J.D. (1985). Learning Logo: What is really learned? Computers in the Schools, 2, 73-77.

Goodman, K. (1986). What's whole in whole language? Portsmouth, NH: Heinemann.

Hawkins, J. & Sheingold, K. (1986). The beginning of a story: computers and the organization of learning in classrooms. In J.A.Culbertson & L.L.Cunningham (Eds.), Microcomputers and education, 85th yearbook of the NSSE (pp. 40-58). Chicago: University of Chicago Press.

Hoffman, R. (1985). Educational software: Evaluation? No! Utility? Yes! Journal of Learning Disabilities, 18, 358-60.

Johanson, R. (1985). School computing: Some factors affecting student performance. Paper presented at the annual meeting of the American Educational Research Association, Chicago, April 1, (ERIC Document Reproduction Service, ED-258-554).

Judd, C.H. (1908). The relation of special training to general intelligence. Educational Review, 36, 28-42.

Kurland, D.M., Mawby, R. & Kahir, N. The development of programming expertise in adults and children. Paper presented at the annual meeting of the American Educational Research Association, New Orleans. (ERIC Document Reproduction Service, ED-257-441).

Kurland, D.M., Pea, R.D., Clement, C. & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. Journal of Educational Computing Research, 2(4), 429-457.

Linn, M.C. (1985). The cognitive consequences of programming instruction in classrooms. Educational Researcher, 14(5), 14-16, 25-29.

Lockheed, M., & Mandinach, E. (1986). Trends in educational computing: Decreasing interest and the changing focus of instruction. Educational Researcher, 15(5), 21-26.

Mandinach, E.B. & Linn, M.C. (1986). The cognitive effects of computer learning environments. Journal of Educational Computing Research, 2(4), 411-427.

Moss, D.S. (1987). Intelligent databases. Byte, 12(1), 97-106.

Nichol, J., Dean, J., & Briggs, J. (1985). Powerful Prolog. The Times Educational Supplement - Extra. Oct. 25, 42.

Nichol, J. & Dean, J. (1984). Pupils, computers and history teaching. In M. Yazdani (Ed.) New horizons in educational computing (pp.190-203). Chichester, England: Ellis Horwood.

Nichol, J., Briggs, J. & Dean, J. (1986). Prolog in Education. PEGBOARD, 2(1), 84-96.

Nickerson, R., Perkins, & Smith (1985). Teaching thinking skills. Hillsdale, NJ: Lawrence Erlbaum Associates

Novak, J.D. (1977). A theory of education. Ithaca, NY: Cornell University Press.

Papert, S. (1980). Mindstorms: Children, computers and powerful ideas. New York: Basic Books.

Papert, S. (1984). Tomorrow's classrooms. In M. Yazdani (Ed.) New horizons in educational computing (pp. 17-20). Chichester, England: Ellis Horwood.

Papert, S. (1987). Computer criticism vs. technocentric thinking. Educational Researcher, 16(1), 22-30.

Patterson, J.H. & Smith, M.S. (1986). The role of computers in higher-order thinking. In J.A.Culbertson & L.L.Cunningham (Eds.), Microcomputers and education, 85th yearbook of the NSSE (pp. 81-108). Chicago: University of Chicago Press.

Pea, R.D. (1984). What will it take to learn thinking skills through computer programming? Paper presented at the annual meeting of the American Educational Research Association, New Orleans. (ERIC Document Reproduction Service, ED-257-441).

Pea, R.D. and Kurland, D.M. (1984). On the cognitive effects of learning computer programming: A critical look. New Ideas in Psychology, 2, 137-168.

Rogers, J.B. (1986). A Prolog primer. Reading, MA: Addison-Wesley.

Smith, D. (1986). How long is a piece of string: Issues in the classroom evaluation of micro-Prolog. PEGBOARD, 2(1), 108-123.

Thorndike, E.L. & Woodworth, R.S. (1901). The influence of improvement in one mental function upon the efficiency of other functions. Psychological Review, 8, 247-61, 384-95, 553-64.

Walker, D.F. (1986). Computers and the curriculum. In J.A.Culbertson & L.L.Cunningham (Eds.), Microcomputers and education, 85th yearbook of the NSSE (pp. 22-39). Chicago: University of Chicago Press.

Walker, D., & Schaffarzick, J. (1974). Comparing curricula. Review of Educational Research, 44, 83-111.

Wittgenstein, L. (1961). Tractatus logico-philosophicus. Pears, D.F. and McGuinness, B.F. (tr.) New York: Humanities Press.

Yazdani, M. (Ed.) (1984). New horizons in educational computing. Chichester, England: Ellis Horwood.